# COSC 3P95- Software Analysis & Testing

# Assignment 1

**Due date**: Monday, Oct 16th, 2023, at **23:59** (11:59 pm)

**Delivery method:** This is an individual assignment. Each student should submit one PDF through Brightspace.

**Attention:** This assignment is worth 10% of the course grade. Please also check the Late Assignment Policy.

**Name: Dominique Mitchell-Williamson Student ID: 6535595**

## Questions:

1- Explain the difference between "sound" and "complete" analysis in software analysis. Then, define what true positive, true negative, false positive, and false negative mean. How would these terms change if the goal of the analysis changes, particularly when "positive" means finding a bug, and then when "positive" means not finding a bug. **(10 pts)**

**A sound analysis focuses on not reporting false positive bugs, never accepts bad programs. While a complete analysis prioritizes detecting all vulnerabilities in a software, never rejects bad programs.**

**True positive – when a tool reports a vulnerability**
**True negative – when a tool reports a non-vulnerability**
**False positive – when a tool repots a non-existent vulnerability**
**False negative – when a tool fails to report a vulnerability**

**If positive means finding a bug the terms would be the same. But if positive means to not find a bug then the true positive and true negative definitions switch while the false positive and false negative switch as well.**

2- Using your preferred programming language, implement a random test case generator for a sorting algorithm program that sorts integers in ascending order. The test case generator should be designed to produce arrays of integers with random lengths, and values for each sorting method.

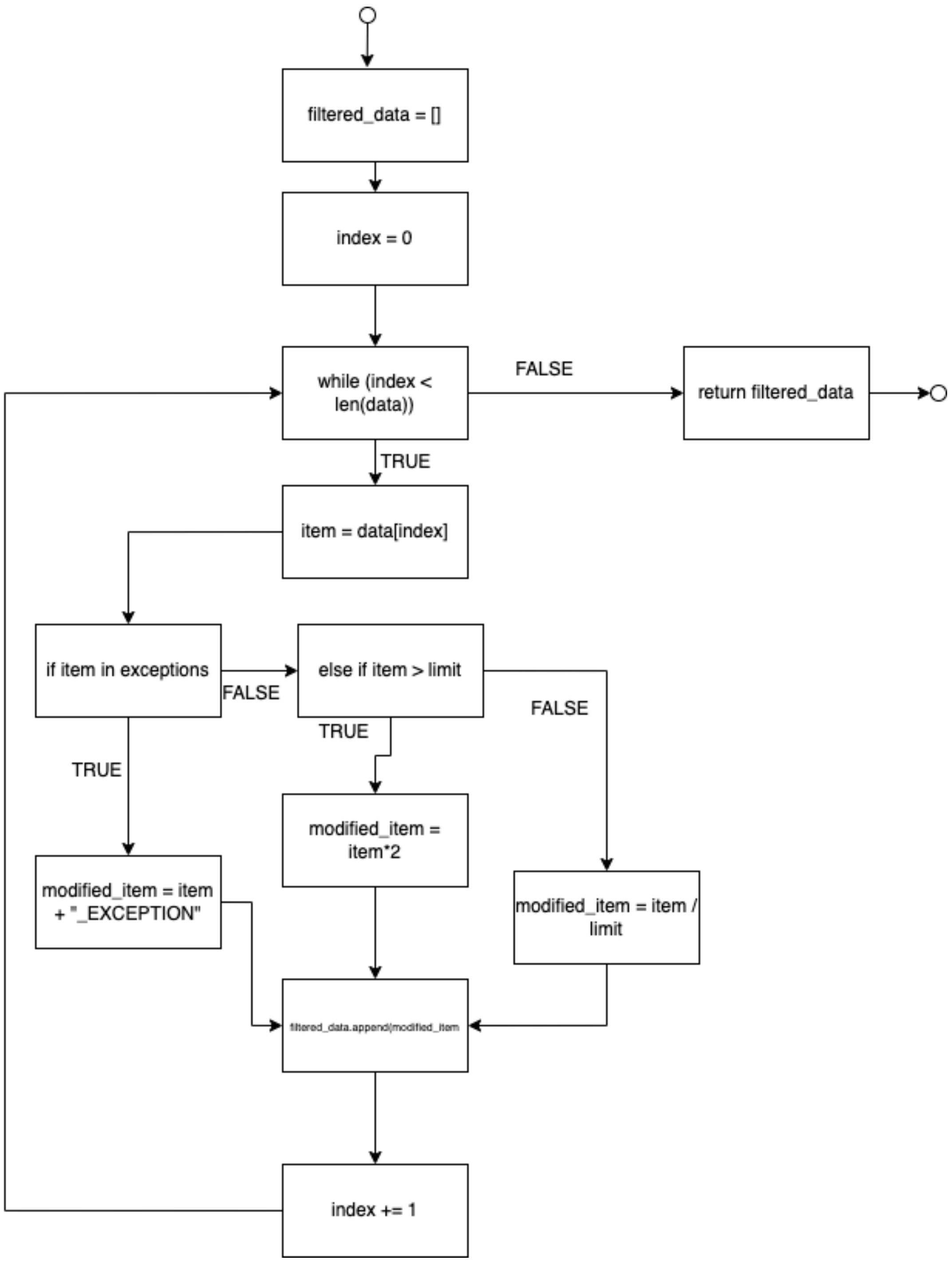
A) Your submission should consist of:

a. Source code files for the sorting algorithm and the random test case generator.
b. Explanation of how your method/approach works and a discussion of the results (for example, if and how the method was able to generate or find any bugs, etc.). You can also include bugs in your code and show your method is able to find the input values causing that.
c. Comments within the code for better understanding of the code.
d. Instructions for compiling and running your code.
e. Logs generated by the print statements, capturing both input array, output arrays for each run of the program.

f. Logs for the random test executions, showing if the test was a pass and the output of the execution (e.g., exception, bug message, etc.).

B) Provide a context-free grammar to generate all the possible test-cases. **(18 + 8 = 26 pts)**

**<test_case> -> “[<test_case>, <integer>] | <integer>**
**<Integer> -> <integer><constant> | <constant>**
**<Constant> -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0**



3- A) For the following code, manually draw a control flow graph to represent its logic and structure.

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

The code is supposed to perform the followings:

a. If an item is in the exceptions list, the function appends "_EXCEPTION" to the item.
b. If an item is greater than a given limit, the function doubles the item.
c. Otherwise, the function divides the item by 2.

**^ Control flow graph above ^**

B) Explain and provide detailed steps for "random testing" the above code. No need to run any code, just present the coding strategy or describe your testing method in detail. **(8 + 8 = 16 pts)**

**I would randomly test the above code by selecting the data, limit, and exceptions parameters differently for each test case. Through this I would watch for expected results and take note of those that didn't meet expectations along the way. The important this is to take note of how the program will handle it's given input, either producing expected or unexpected results then taking note.**

4- A) Develop 4 distinct test cases to test the above code, with code coverage ranging from 30% to 100%. For each test-case calculate and mention its code coverage.

a. **filterData([3,6,9], 2, [])**
    i. **9 lines passed / 13 lines of code = 69% code coverage**
b. **filterData([7, 8, 23, 17], 20, [1,2,6,8])**
    i. **13 lines passed / 13 lines of code = 100% code coverage**
c. **filterData([], 5, [])**
    i. **4 lines passed / 13 lines of code = 31% code coverage**
d. **filterData([20,25,30,40, 120], 100, [24,56,89])**
    i. **11 lines passed / 13 lines of code = 85% code coverage**

B) Generate 6 modified (mutated) versions of the above code.

```
def mutate1data(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index >= len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

```
def mutate2Data(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item < limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
def mutate3Data(data, limit, exceptions):
    filtered_data = []
    index = 3
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

```
def mutate4Data(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item - limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
def mutate5Data(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 3

    return filtered_data
def mutate6Data(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 5
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

C) Assess the effectiveness of the test cases from part A by using mutation analysis in conjunction with the mutated codes from part B. Rank the test-cases and explain your answer.

- **Original**
  - **filterData([3,6,9], 2, []) -> [6,12,18]**
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> [(7/20), 8_EXCEPTION, 46, (17/20)]**
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40,120], 100, [24,56,89]) -> [(20/100), (25/100), (30/100), (40/100), 240]**

- **Mutate1Data**
  - **filterData([3,6,9], 2, []) -> []***
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> []***
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40, 120], 100, [24,56,89]) -> []***

- **Mutate2Data**
  - **filterData([3,6,9], 2, []) -> [(3/2), (6/2), (9/2)]***
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> [14,8_EXCEPTION, (23/20), 34]***
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40, 120], 100, [24,56,89]) -> [40,50,60,80,(120/100)]***

- **Mutate3Data**
  - **filterData([3,6,9], 2, []) -> []***
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> [(17/20)]***
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40, 120], 100, [24,56,89]) -> [(30/100), (40/100), 240]***

- **Mutate4Data**
  - **filterData([3,6,9], 2, []) -> [6, 12, 18]**
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> [-13, 8_EXCEPTION, 46, -3]***
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40, 120], 100, [24,56,89]) -> [-80, -75, -70, -60, 240]***

- **Mutate5Data**
  - **filterData([3,6,9], 2, []) -> [6]***
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> [(7/20), (17/20)]***
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40, 120], 100, [24,56,89]) -> [(20/100), (40/100)]***

- **Mutate6Data**
  - **filterData([3,6,9], 2, []) -> [15, 30, 45]***
  - **filterData([7, 8, 23, 17], 20, [1,2,6,8]) -> [(7/20), 8_EXCEPTION, 115, (17/20)]***
  - **filterData([], 5, []) -> []**
  - **filterData([20,25,30,40, 120], 100, [24,56,89]) -> [(20/100), (25/100), (30/100), (40/100), 600]***

    **TEST CASE MUTATION ANALYSIS: based off mutation score (higher percentage -> greater rating for test case)**

    **1- 5 mutants killed / 6 total mutants = 83% mutation score (3rd)**
    **2- 6 mutants killed / 6 total mutants = 100% (1st / 2nd)**
    **3- 0 mutants killed / 6 total mutants = 0% (4th)**
    **4- 6 mutants killed / 6 total mutants = 100% (1st / 2nd)**

D) Discuss how you would use path, branch, and statement static analysis to evaluate/analyse the above code. **(4 * 8 = 32 pts)**

**Path**

- **I would focus on the logical decisions the program makes. For this analysis I'd create test cases that would go through each and logical decision point different than other paths. These iterations would then be reviewed and analyzed.**

**Branch**

- **I would check to make sure each conditional branch functions as intended, making input for the condition matches requirements and specification**

**Statement static**

- **This would entail finding the percentage of code encounter. Found by taking the (lines of code encounter / total lines of code) = percentage encountered. This would help with statement static analysis.**

5- The code snippet below aims to switch uppercase characters to their lowercase counterparts and vice versa. Numeric characters are supposed to remain unchanged. The function contains at least one known bug that results in incorrect output for specific inputs.

```
def processString(input_str):
    output_str = ""
    for char in input_str:
        if char.isupper():
            output_str += char.lower()
        elif char.isnumeric():
            output_str += char * 2
        else:
            output_str += char.upper()

    return output_str
```

In this assignment, your tasks are:

a. Identify the bug(s) in the code. You can either manually review the code (a form of static analysis) or run it with diverse input values (a form of manual random testing). If you are unable to pinpoint the bug using these methods, you may utilize a random testing tool or implement random test case generator in code. Provide a detailed explanation of the bug, identify the line of code causing it, and describe your strategy for finding it.

   - Highlighted points out number characters is changed, when it's supposed to be unchanged
     - I would remove the "* 2" to fix this bug
   - A space or special character may break this code
     - In the event these characters show up I would handle them similarly to a numerical character should

b. Implement Delta Debugging, in your preferred programming language to minimize the input string that reveals the bug. Test your Delta Debugging code for the following input values provided.
   i. “abcdefG1”
   ii. “CCDDEExy”
   iii. “1234567b”
   iv. “8665”

Briefly explain your delta-debugging algorithm and its implementation and provide the source code in/with your assignment. **(4 + 12 = 16 pts)**

6- Extra Credit Assignment: Create a GitHub repository to host all the elements of this assignment. This includes source codes, test data, and any screenshots or logs you have generated. Submit the GitHub link along with your main submission through Brightspace. **(5 pts)**

## Marking Scheme:

*Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Lack of clarity may lead you to lose marks, so keep it simple and clear.*

***Submission:***

*The submission is expected to contain a sole word-processed document. The document can be in either **DOC or PDF** format; it should be a single column, at least single-spaced, and at least in font 11. It is strongly recommended to use the assignment questions to facilitate marking: answer the questions just below them for easier future reference.*

***Late Assignment Policy:***

*A one-time penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, four days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor’s discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor’s discretion.*

***Plagiarism:***

*Students are expected to respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee’s attention. In case plagiarism is determined, the activity will be canceled, and the author(s) will be subject to university regulations. For further information on this sensitive subject, please refer to the document below: **https://brocku.ca/node/10909***

```
# python code for question 2
# from terminal be in the correct working directory as question2.py
# ensure python3 is installed (up-to-date) before running
# invoke "python3 question2.py"
# this code will initially produce 10 iterations of random list generation
# which will be organized in ascending order using insertion sort

# import and initialize random instance with given student number seed
import random
seed = 5550
random.seed(seed)
print("Seed: " + str(seed) + "\n")


# procedure to replicate random test case generation with insertion sort algorithm
def testing():
    total_tests = 10    # total_tests can be modified to desired number of tests
    for x in range(1, total_tests):
        insertion_algo()
        print("ITERATION PASSED")

    print("\ntesting finished")


# insertion sort procedure to organize list produced from random_test_gen()
def insertion_algo():
    arg = random_test_gen()     # copies random list to arg variable

    print("Executing insertion sort")
    # loop to iterate through each item in list
    for i in range(1, len(arg)):
        key = arg[i]    # copy value of current position to variable key
        j = i - 1       # copy position of key'd value to variable j

        # loop through sorted list until end reached or key > sorted position in list
        while j >= 0 and arg[j] > key:
            arg[j + 1] = arg[j]     # shifts value of sorted item right 1 cell
            j -= 1      # decrement to smaller item in list

        arg[j + 1] = key        # position for key placed in correct position

    print(arg)


# function to generate random list of integers ranging from mini-maxi
def random_test_gen():
    mini = 1
    maxi = 20
    length = random.randint(mini, maxi)

    print("\nInitializing list of " + str(length) + " items")
    array = [0]*length
    print(array)

    print("Filled with random integers from 1-" + str(maxi))
    for x in range(length):
        array[x] = random.randint(mini, maxi)
    print(array)
    return array


# execution of random generation and insertion sort algorithm
testing()
```